

Symbolic Learning for Adaptive Agents

John Lloyd

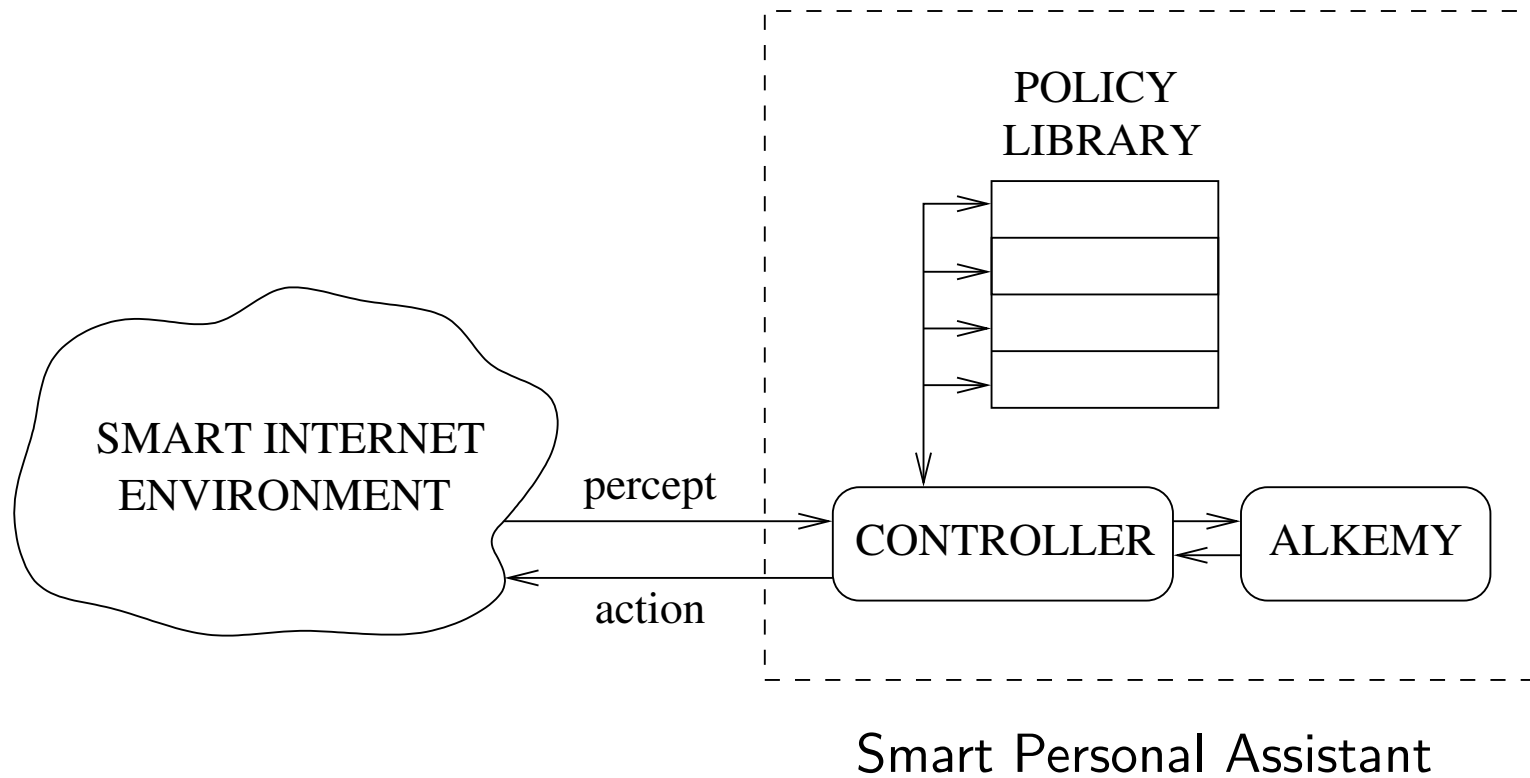
Computer Sciences Laboratory
The Australian National University



Overview

- Symbolic machine learning for adaptive agents
- Predicate construction in higher-order logic
- An illustration of the ideas
- Summary

Agent Architecture



Research Issues

Seamless integration of reasoning and learning capabilities of agents

One approach: integration of BDI agents and symbolic machine learning

The work here was originally inspired by that of Džeroski, De Raedt, Driessens and Blockeel on relational reinforcement learning

In this talk, I concentrate on the issue of *adaptivity*:

- the agent should be able to adapt to a changing environment
- the agent should be able to adapt to changing tasks

Approach to Adaptivity

A *policy* is a function

$$\pi : State \rightarrow Action$$

For convenience, we put this into relational form:

$$policy : State \times Action \rightarrow \Omega$$

This suggest an on-line learning approach to adaptivity:

- policy defined by a decision tree
- an hypothesis language in which the definition of the policy can vary
- a source of training examples to guide the change of policy

‘Logic for Learning’

- Uses a polymorphic higher-order logic (based on Church’s simple theory of types)
- Develops a set of terms (called *basic* terms) for representing individuals (includes sets, multisets, ...)
- Shows how to use the logic to build hypothesis languages
- Shows how to use the logic for computation, and provides a foundation for functional logic programming
- Provides a higher-order foundation for logic programming and inductive logic programming

Transformations

Predicates are built up by composing transformations, which are defined as follows.

Definition. A *transformation* f is a function having a signature of the form

$$f : (\varrho_1 \rightarrow \Omega) \rightarrow \cdots \rightarrow (\varrho_k \rightarrow \Omega) \rightarrow \mu \rightarrow \sigma,$$

where any parameters in $\varrho_1, \dots, \varrho_k$ and σ appear in μ , and $k \geq 0$.

The type μ is distinguished and is called the *source* of the transformation, while the type σ is called the *target* of the transformation.

The number k is called the *rank* of the transformation.

Note that composition and transformations are higher-order.

Transformations 2

Example. The transformation

$$\wedge_n : (a \rightarrow \Omega) \rightarrow \cdots \rightarrow (a \rightarrow \Omega) \rightarrow a \rightarrow \Omega$$

defined by

$$\wedge_n p_1 \dots p_n x = (p_1 x) \wedge \cdots \wedge (p_n x),$$

where $n \geq 2$, provides a ‘conjunction’ with n conjuncts.

Example. The transformation $top : a \rightarrow \Omega$ is defined by $top x = \top$, for each x .

Standard Predicates

Transformations are used to define a particular class of predicates, called standard predicates, that are used in hypothesis languages.

Definition. A *standard predicate* is a term of the form

$$(f_1 p_{1,1} \dots p_{1,k_1}) \circ \dots \circ (f_n p_{n,1} \dots p_{n,k_n}),$$

where f_i is a transformation of rank k_i ($i = 1, \dots, n$), the target of f_n is Ω , p_{i,j_i} is a standard predicate ($i = 1, \dots, n$, $j_i = 1, \dots, k_i$), $k_i \geq 0$ ($i = 1, \dots, n$) and $n \geq 1$.

Example. If p , q , and r are standard predicates (having appropriate type) and $\neg : \Omega \rightarrow \Omega$ is negation, then $(\wedge_3 p q r) \circ \neg$ is a standard predicate.

Predicate Rewrite Systems

Definition. A *predicate rewrite system* is a finite relation \rhd on the set of standard predicates satisfying the following two properties.

1. For each $p \rhd q$, the type of p is more general than the type of q .
2. For each $p \rhd q$, there does not exist $s \rhd t$ such that q is a proper subterm of s .

If $p \rhd q$, then $p \rhd q$ is called a *predicate rewrite*, p the *head*, and q the *body* of the predicate rewrite.

Predicate Rewrite Systems 2

Definition. A subterm of a standard predicate $(f_1 p_{1,1} \dots p_{1,k_1}) \circ \dots \circ (f_n p_{n,1} \dots p_{n,k_n})$ is *eligible* if it is a suffix of the standard predicate or it is an eligible subterm of p_{i,j_i} , for some $i \in \{1, \dots, n\}$ and $j_i \in \{1, \dots, k_i\}$.

Definition. Let \succrightarrow be a predicate rewrite system and p a standard predicate. An eligible subterm r of p is a *redex* with respect to \succrightarrow if there exists a predicate rewrite $r \succrightarrow b$ such that the type of b and the positional type of r in p are unifiable. In this case, r is said to be a redex *via* $r \succrightarrow b$.

Definition. Let \succrightarrow be a predicate rewrite system, and p and q standard predicates. Then q is obtained by a *predicate derivation step* from p using \succrightarrow if there is a redex r via $r \succrightarrow b$ in p and $q = p[r/b]$. The redex r is called the *selected* redex.

Illustration in Blocks World

$B_0, B_1, B_2, B_3, B_4, B_5, B_6, B_7, B_8, B_9, Floor : Object$

$Stack = List\ Object$

$World = \{Stack\}$

$OnState = Object \times Object$

$Intention = \{OnState\}$

$Action = Object \times Stack$

$State = World \times Intention$

$Individual = State \times Action.$

The logic provides a rich class of terms, including sets and multisets, for representing individuals. (These are the *basic* terms.)

Domain-specific Transformations

$$\text{extractAction} : \text{Individual} \rightarrow \text{Object} \times \text{Stack} \times \text{Individual}$$

takes an individual as input and returns the triple consisting of the block in the action, the stack in the action, and the individual.

$$\text{projB} : \text{Object} \times \text{Stack} \times \text{Individual} \rightarrow \text{Stack} \times \text{Individual}$$

takes as input a triple consisting of a block, a stack, and an individual, and returns the pair consisting of the stack and the individual.

$$\text{isConstructive} : \text{Object} \times \text{Stack} \times \text{Individual} \rightarrow \Omega$$

takes as input a triple consisting of a block, a stack, and an individual, and returns true if moving the block to the stack in the world component of the individual is constructive; and returns false, otherwise.

Predicate Rewrite System for Policies

$top \rightsquigarrow \wedge_3 top\ top\ top$

$top \rightsquigarrow extractAction \circ projA \circ top$

$top \rightsquigarrow extractAction \circ projB \circ top$

$top \rightsquigarrow extractAction \circ top$

$top \rightsquigarrow isDeadlocked$

$top \rightsquigarrow isMisplaced$

$top \rightsquigarrow isFloor$

$top \rightsquigarrow isConstructive.$

Policies

The *simple* policy either makes a constructive move or else moves a block to the floor.

Here is the *simple* policy as an ALKEMY decision tree.

$$\begin{aligned} \text{policy}_{\text{simple}} x = & \\ & \text{if } \text{extractAction} \circ \text{isConstructive } x \\ & \text{then } \top \\ & \text{else if } \text{extractAction} \circ \text{projB} \circ \text{isFloor } x \\ & \text{then } \top \\ & \text{else } \perp. \end{aligned}$$

Policies 2

The *GN1* policy is as follows: if there is a constructive move, then do it; else arbitrarily choose a misplaced block and move it to the floor.

Here is the *GN1* policy as an ALKEMY decision tree.

$$\begin{aligned} & \textit{policy}_{GN1} \ x = \\ & \quad \textit{if } \textit{extractAction} \circ \textit{isConstructive} \ x \\ & \quad \textit{then } \top \\ & \quad \textit{else if } \wedge_3 (\textit{extractAction} \circ \textit{projA} \circ \textit{isMisplaced}) \\ & \quad \quad (\textit{extractAction} \circ \textit{projB} \circ \textit{isFloor}) \\ & \quad \quad (\textit{isDeadlocked}) \ x \\ & \quad \textit{then } \top \\ & \quad \textit{else } \perp. \end{aligned}$$

Training Examples for Policy Adaptation

Where do the training examples come from?

This depends on the application. For example, they may come from user interaction. Alternatively, they may come from a (model-free) reinforcement learning method such as Q learning.

The Q function has signature

$$Q : State \times Action \rightarrow \mathbb{R}.$$

Knowing Q allows one to calculate an optimal action:

$$\pi^*(s) = \arg \max_a Q(s, a).$$

Thus Q can provide a stream of training examples for the policy.

Approximating the Q Function

The Q function is determined, roughly speaking, by the length of the shortest path from the current state to the nearest goal.

Thus the Q function has a very different hypothesis language to the policies, depending instead on a transformation *estimatedPathLength* that approximates the path length.

ALKEMY incrementally builds a regression tree that approximates the Q function of the following form:

$$\begin{aligned} Q\ x = & \\ & \text{if } \text{estimatedPathLength} \circ (= 0)\ x \text{ then } q_0 \\ & \text{else if } \text{estimatedPathLength} \circ (= 1)\ x \text{ then } q_1 \\ & \vdots \end{aligned}$$

An Experiment

The blocks world initially contains 5 blocks.

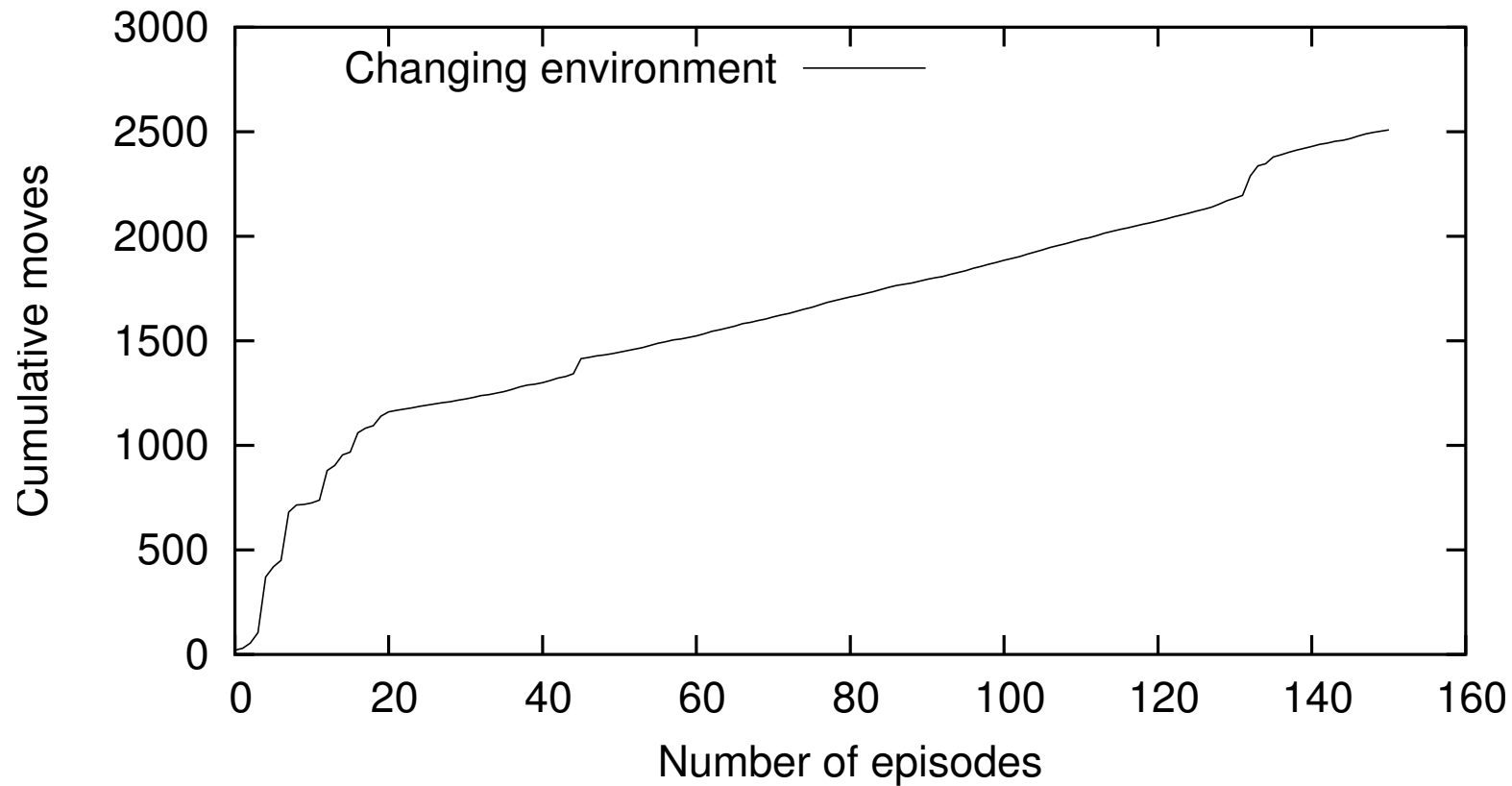
At episode 30, a new block is added to the environment and, at episode 60, yet another block is added.

The task of the agent is also incremented in the richer environments from sets of 5 on-states to sets of 6 on-states, and finally to 7 on-states in a 7 blocks world.

The agent is initialised with *policy_{simple}*.

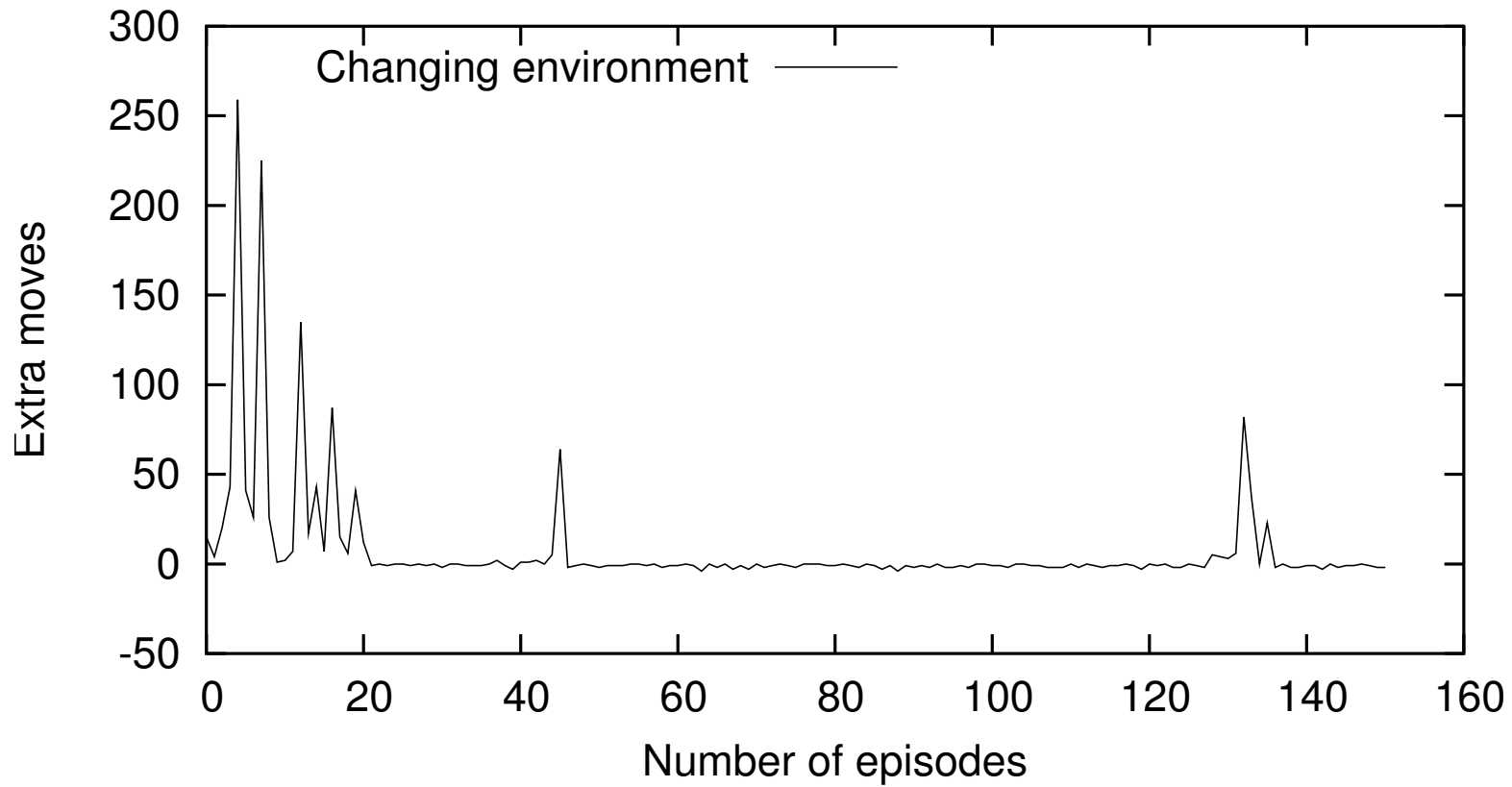
Result

5 blocks, 5 intentions to 7 blocks, 7 intentions



Result 2

5 blocks, 5 intentions to 7 blocks, 7 intentions



Summary

- The setting is a polymorphically typed, higher-order logic
- There is a rich class of terms to model individuals
- Predicate rewrite systems give precise and explicit control over hypothesis languages
- There is a computational mechanism for ‘evaluating’ terms
- *ALKEMY* is a symbolic classification and regression learning system that can be used for batch or on-line learning
- We are currently working on several tasks for the smart personal assistant, such as a personal news service, that would benefit greatly from personalisation via adaptation

Pointers and Acknowledgements

J.W. Lloyd, 'Logic for Learning', Cognitive Technologies, Springer, 2003

ALKEMY is available at <http://csl.anu.edu.au/~kee/Alkemy>

A working paper on the agents work is available from me

Many thanks to the following for help on 'Logic for Learning':

- Antony Bowers, Peter Flach, Thomas Gärtner, Christophe Giraud-Carrier, Kee Siong Ng

and to those in the SITCRC agents group at ANU:

- Robert Bridle, Joshua Cole, Eric McCreath, Mathi Nagarajan, Kee Siong Ng